NASA Contractor Report 178274

ICASE REPORT NO. 87-21

ICASE

ANALYSIS OF A PARALLELIZED NONLINEAR ELLIPTIC BOUNDARY VALUE PROBLEM SOLVER WITH APPLICATION TO REACTING FLOWS

David E. Keyes

Mitchell D. Smooke

Contract No. NAS1-18107

April 1987

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia 23665

Operated by the Universities Space Research Association

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665

Analysis of a Parallelized Nonlinear Elliptic Boundary Value Problem Solver with Application to Reacting Flows

David E. Keyes†
Department of Mechanical Engineering, Yale University and
Institute for Computer Applications in Science and Engineering

and
Mitchell D. Smooke
Department of Mechanical Engineering, Yale University

Abstract: A parallelized finite difference code based on Newton's method for systems of nonlinear elliptic boundary value problems in two dimensions is analyzed in terms of computational complexity and parallel efficiency. An approximate cost function depending on 15 dimensionless parameters (including discrete problem dimensions, convergence parameters, and machine characteristics) is derived for algorithms based on stripwise and boxwise decompositions of the domain and a one-to-one assignment of the strip or box subdomains to processors. The sensitivity of the cost function to the parameters is explored in regions of parameter space corresponding to model small-order systems with inexpensive function evaluations and also a coupled system of nineteen equations with very expensive function evaluations (a reacting flow model of engineering interest which motivates the work). The algorithm has been implemented on the Intel Hypercube, and some experimental results for the model problems with stripwise decompositions are presented and compared with the theory. In the context of computational combustion problems, multiprocessors of either message-passing or shared-memory type may be employed with stripwise decompositions to realize speedups of $O(n)$, where n is mesh resolution in one direction, for reasonable n. To realize speedups of $O(n^2)$, the total number of mesh points, only hypercubes appear attractive. These results must be qualified by hardware assumptions, including sufficient local memory per processor to hold all of the data defined on the associated subdomain, and selection of machine parameters typical of presently commercially available components.

† This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-18107 and AFOSR 85-0189 while the first author was in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665.

1. Introduction

Systems of nonlinear elliptic boundary value problems (BVPs) constitute an important class of problems in large-scale scientific computation. In many instances, including the example considered herein of the detailed modeling of steady multidimensional reacting flows, evaluation of the governing equation residuals at a given solution iterate constitutes a significant expense, so methods which make efficient use of the function evaluations are required. For such problems, robust variations of Newton's method are often preferable to less fully coupled iterative methods or associated time-marching methods (see, *e.g.*, [12]), if memory capacity allows, as we tacitly assume it does throughout this paper.

In contrast, for many other systems of nonlinear elliptic BVPs, such as the steady incompressible Navier-Stokes problem, the cost of a residual function evaluation is much less than the cost of forming and solving the large linear systems arising at each stage of Newton's method, so that various time or time-like relaxation methods requiring less implicitness have traditionally been preferred. Explicit methods generally parallelize very efficiently, and might thus appear to be superior to more implicit methods as parallel algorithms, even where this might not be true in the serial context. We demonstrate through complexity estimates that there is a wide range of parameter space over which Newton-based methods also parallelize within acceptable ranges of computational efficiency, and that this region includes reacting flow problems of current interest.

The computational complexity model is introduced in general terms, but is ultimately made problem-specific for the purpose of reducing the 15 parameters of the model to a manageable number of interesting ones. Some of these parameters are unavailable or impractical to obtain from theory alone, and are supplied empirically from actual serial runtimes. Parallel numerical experiments for realistic reacting flows are as yet impractical on the hardware conveniently available to the authors for this study, but results on model problems in representative ranges for the parameters n, the resolution of the grid, and p, the number of processors employed, support various testable results of the complexity analysis.

The organization of this paper is as follows. In section 2 we sketch in general terms a modified Newton algorithm for the solution of nonlinear elliptic boundary value problems by finite discretization methods. Section 3 describes a serial implementation of this algorithm which has recently been applied successfully to the computation of an axisymmetric, over-ventilated, subsonic, laminar methane-air jet diffusion flame. Parallel implementation issues and a complexity theory are presented in section 4. Section 5 contains some actual performance data for model systems obtained on the Intel Hypercube, and a discussion of its implications for modeling realistic systems. We conclude in section 6 with some recommendations for the further development of the parallel computation of reacting flows.

2. An Algorithm for Nonlinear Elliptic Boundary Value Problems

2.1. Newton's Method for a Generic Nonlinear System

We consider in this section a discretized elliptic system in the generic form

$$F(\phi, \chi, \pi) = 0, \tag{2.1}$$

where ϕ is a vector of unknowns, χ and π are vectors of parameters, and F is a vector-valued function in which the components of ϕ, χ, and π may enter nonlinearly. For the sake of definiteness in the following discussion, we assume that F arises from a low-order finite difference discretization on a tensor-product grid in two dimensions, with m gridpoints in one direction and n in the other, and that there are r unknowns defined at each gridpoint. The unknown vector is then comprised of rmn elements, including boundary values, conveniently enumerated by triple subscripts as ϕ_{kij}

for $k = 1,\ldots,r; i = 1,\ldots,m; j = 1,\ldots,n$, and F is likewise comprised of rmn scalar equations, including discrete boundary conditions. The vector of parameters χ consists of s gridfunctions which are not necessarily independent of the unknowns ϕ, but are in general given by a set of explicit algebraic relationships which involve only unknowns defined at a single point of the form

$$\chi_{lij} = \tilde{\chi}_l(\phi_{1ij},\ldots,\phi_{rij})$$

for $l = 1,\ldots,s; i = 1,\ldots,m; j = 1,\ldots,n$.

The reason for distinguishing between the ϕ and χ gridfunctions is that the equations for the latter contain no spatial derivatives and what would otherwise be a system of order $(r+s)mn$ can be condensed algebraically with modest effort to a system of order rmn. The parameters π are global parameters which are independent of the unknowns ϕ. As an illustration of this partitioning in the modeling of reacting flows, the local mass fraction of a species belongs in ϕ, its local coefficient of diffusion in the gas mixture belongs in χ, and the activation energy of its formation in a particular elementary reaction belongs in π. The parameters χ and π will be suppressed in what follows, but their presence is felt through the terms involving s in the complexity model in section 4.

The system (2.1) may be solved efficiently by a damped modified Newton method provided that an initial iterate $\phi^{(0)}$ sufficiently close to the solution $\phi^\star$ is supplied. The damped modified Newton iteration is given by

$$\phi^{(k+1)} = \phi^{(k)} + \lambda^{(k)}\delta\phi^{(k)}, \tag{2.2}$$

where

$$\delta\phi^{(k)} = (\tilde{J}^{(k)})^{-1}F(\phi^{(k)}), \tag{2.3}$$

where the matrix $\tilde{J}^{(k)}$ is an approximation to the actual Jacobian matrix evaluated at the k^{th} iterate. We refer to $\delta\phi^{(k)}$ as the k^{th} update. When $\lambda^{(k)} = 1$ and $\tilde{J}^{(k)} = J^{(k)} \equiv \frac{\partial F}{\partial \phi}(\phi^{(k)})$, for all k, a pure Newton method is obtained.

For the pure Newton method, the iteration (2.2) possesses at least a quadratic rate of convergence when the hypotheses of the Kantorovich theorem are satisfied. That is, $||\phi^{(k+1)} - \phi^\star|| \leq c||\phi^{(k)} - \phi^\star||^2$ for some constant c. For the modified Newton method without restarting (*i.e.*, with $\tilde{J}^{(k)} = \frac{\partial F}{\partial \phi}(\phi^{(0)})$ for all $n \geq 0$), the rate of convergence is not guaranteed to be better than linear. Consequently, there is a tradeoff to be made between the higher cost per iteration of the pure Newton method and the higher iteration count of a modified Newton method. This tradeoff has been variously resolved in the literature. We follow the practice of restarting our modified Newton method whenever the norm of the current update divided by the norm of the first update which was obtained with the Jacobian in current use fails to satisfy a bound derived from the Kantorovich theorem. This bound depends only on the index of the current update relative to that of the last restart (for details, see [13]).

When the size of the successive modified Newton steps is decreasing in accord with the Kantorovich theorem, no damping is necessary ($\lambda^{(k)} = 1$). Since there is no practical means for ensuring that the initial iterate lies within the domain of convergence defined by the theorem, damping is often necessary in the early stages to prevent divergence. In our implementation, we damp $\delta\phi^{(k)}$ for either of two reasons. First, if bounds are specified by the user for certain components of ϕ, $\lambda^{(k)}$ is chosen as the minimum of 1 and of the value which would drive any component of $\phi^{(k+1)}$ to its bound. Second, if the projected *next* update based on damping with the provisional $\lambda^{(k)}$,

$$\delta\phi^{(k+1)} = (\tilde{J}^{(k)})^{-1}F(\phi^{(k)} + \lambda^{(k)}\delta\phi^{(k)}),$$

is larger than the current update $\delta\phi^{(k)}$, then $\lambda^{(k)}$ is recursively halved until this condition no longer holds, or until $\lambda^{(k)}$ becomes smaller than some tolerance. In the latter case, the algorithm unsuccessfully damps to death, and a restart with a better initial condition is recommended.

2.2. A Pseudo-transient Continuation Procedure

A better initial condition for Newton's method is usually obtainable by an elementary continuation procedure, namely driving a pseudo-transient form of (2.1),

$$D\frac{\partial \phi}{\partial t} + F(\phi) = 0, \tag{2.4}$$

where D is a scaling matrix, part of the way towards its steady state [15]. If (2.4) is implicitly time-differenced with the backward Euler method, using time step Δt from a given initial state $\bar{\phi}$ a new system,

$$\bar{F}(\phi) \equiv D\frac{(\phi - \bar{\phi})}{\Delta t} + F(\phi) = 0, \tag{2.5}$$

results, which possesses the Jacobian

$$\bar{J} \equiv \frac{D}{\Delta t} + \frac{\partial F}{\partial \phi}.$$

With an appropriate D and a sufficiently small Δt, (2.5) may in principal be iterated under the successive substitution $\bar{\phi} \leftarrow \phi$ as a time-accurate solution of a physically transient problem. This is generally infeasible in terms of execution time, and possibly unproductive because of spatial truncation errors which remain finite even as the time step becomes infinitesimal. (The form of the algorithm we describe in this paper leaves out consideration of spatial grid adaptivity. This feature is important in combustion problems and has been implemented in the serial version of the code, but not yet in the parallel version, where load balancing considerations are necessitated.)

For a succession of time steps which may be larger than what would be required for accurate resolution of the physical transient, the iteration of (2.5) is a numerically robust procedure for generating an initial iterate for Newton's method. Moreover, due to the close relationship of (2.5) to (2.1), only minor modifications of the steady-state code for (2.1) are required to make it a transient/steady-state hybrid. Adaptive control of the size of the time steps used in (2.5), can enhance the efficiency of such a hybrid algorithm. One useful strategy is to choose Δt based on the temporal truncation error of the most rapidly varying component of the solution. As ϕ approaches $\phi^\star$ and Δt becomes large, this effects a smooth transition to the (infinite time step) steady-state formulation. Alternatively, the growth of Δt beyond a certain Δt_{max} can be used to trigger automatically a transition to the steady-state formulation.

2.3. Algorithmic Implementation of the Hybrid Solver

The Appendix contains two procedures, named **SOLVER** and **NEWTON**, which represent an implementation of the ideas of this section. **SOLVER** operates in either of two modes: a direct treatment of the steady-state problem, or the steady-state problem preceded by one or more adaptive time steps. It has five arguments: the name of a user-supplied subprocedure **F** (simply passed to the internal procedure **NEWTON**) which evaluates the transient or steady-state residual, depending upon the mode; a real solution vector $\tilde{x}$ for communicating the initial iterate upon invocation and the converged solution upon a successful return; an integer n_{time} for setting the mode (if $n_{time} > 0$, time-stepping is attempted for up to n_{time} adaptively chosen steps); an initial time step Δt_1, which is ignored if $n_{time} = 0$; and a flag, $flag_{SOLVER}$, to indicate the state of the procedure upon return. There are two other flags in **SOLVER** which function as logical control parameters: $flag_{NEWTON}$, which communicates the state of the called procedure **NEWTON** upon its return, and $flag_{JACOBIAN}$, which determines the frequency at which the Jacobian is updated. The finest level of control over the Jacobian update frequency lies inside of **NEWTON** itself, but at a coarse level **SOLVER** forces a refreshing of the Jacobian at the outset of a calculation in either mode and also after an unsuccessful time step. There are also two real control parameters internal to

SOLVER: a minimum time step Δt_{min} and a maximum time step Δt_{max}. During normal execution, the time step Δt is adjusted upward or downward according to the progress of the iterations in a subprocedure which may be problem-specific and is regarded as a black box for purposes of this algorithm. Δt_{min} is a lower bound on the size of the time step to keep the computation out of inefficient realms. If the procedure attempts to set a time step smaller than Δt_{min} it is terminated with a "damped to death in transient mode" message. Recovery from this termination calls for a new initial iterate or a willingness to expend lots of CPU time, as indicated by a new, lower, input value of Δt_{min}. If the procedure attempts to set a time step greater than Δt_{max}, direct consideration of the steady-state problem is deemed feasible, and the mode is automatically switched. The steady-state mode is also switched in by default after the number of time steps specified by n_{time} is exhausted. A failure during the steady-state computation is indicated with the flag "damped to death in steady-state mode".

The procedure named **NEWTON** is driven by procedure **SOLVER**. The mode distinctions of **SOLVER** are invisible to **NEWTON**; only the meaning of the argument **F** changes with mode, not the way **F** is handled within **NEWTON** . The procedure has seven arguments: the name of a user-supplied procedure **F** which it calls to evaluate a transient or steady-state residual; a real solution vector $\tilde{x}$ for communicating the initial iterate for the steady or transient **F** upon invocation and the converged solution upon return; an integer n_{time}, a real solution vector from the previous time step $x^{(k-1)}$, and a current time step Δt_k, all three of which are simply passed through **NEWTON** to **F**; a flag $flag_{NEWTON}$ to indicate the state of the procedure upon return, and a flag $flag_{JACOBIAN}$ to control the frequency of updating of the Jacobian. The internal parameter m counts the number of iterations since the current Jacobian was last updated and serves as an index into a table of convergence bounds from [13], denoted α_m below. There are three internal control parameters, as follows. The parameter $n_{modified}$ is the maximum number of iterations between Jacobian updates even if the modified iterations are making their theoretically anticipated progress. The parameter λ_{min} is a lower bound on the size of the adaptively chosen damping parameter, which is designed to abort a calculation started outside of the domain of convergence. Finally, $\xi_{converge}$ is an absolute convergence tolerance, chosen with due consideration to the scaling of the components of x and to the machine precision. Note that convergence of **NEWTON** is based on the (theoretically motivated) norm of the update, not on the norm of the residual itself. The size of the residual at convergence depends on the condition number of the Jacobian matrix, and may be quite large in detailed kinetics combustion problems even after convergence of the solution to machine precision.

The level of algorithmic detail furnished in the Appendix is sufficient to identify the five basic tasks which together account for almost all of the execution time required by the code. In order of increasing importance for reacting flow calculations, they are: (1) DAXPY vector arithmetic, (2) the evaluation of norms, (3) the solution of linear equations involving the Jacobian matrix, (4) the evaluation of governing equation residuals, and (5) the evaluation of Jacobians.

3. The Physical Model and a Serial Implementation

3.1. The Continuous Governing System

The form of the governing equations determines to a large extent the choice of method for the linear algebra and the means for evaluating the Jacobian. For definiteness, we cite without derivation the following set of elliptic boundary value problems (see, *e.g.*,[3]). Though typical of many reacting flow models, they do not include effects which will in some contexts be essential (*e.g.*, radiation, turbulence), and which could drastically alter the algebraic form of the governing system. The problem motivating the present work is a two-dimensional axisymmetric isobaric laminar diffusion flame.

Let r and z denote the radial and axial directions, v_r and v_z the respective velocity components,

and ρ the density. We introduce the compressible Stokes streamfunction ψ such that

$$\rho r v_r = -\frac{\partial \psi}{\partial z} \quad \text{and} \quad \rho r v_z = \frac{\partial \psi}{\partial r},$$

and the vorticity

$$\omega = \frac{\partial v_r}{\partial z} - \frac{\partial v_z}{\partial r}.$$

The species concentrations by mass are Y_k, for $k = 1, 2, \ldots, K$, where K is the total number of species in the reaction mechanism. The temperature is T. These principal fields satisfy the following equations.

Streamfunction:

$$\frac{\partial}{\partial z}\left(\frac{1}{r\rho}\frac{\partial \psi}{\partial z}\right) + \frac{\partial}{\partial r}\left(\frac{1}{r\rho}\frac{\partial \psi}{\partial r}\right) + \omega = 0. \tag{3.1}$$

Vorticity:

$$r^2\left[\frac{\partial}{\partial z}\left(\frac{\omega}{r}\frac{\partial \psi}{\partial r}\right) - \frac{\partial}{\partial r}\left(\frac{\omega}{r}\frac{\partial \psi}{\partial z}\right)\right] - \frac{\partial}{\partial r}\left(r^3\frac{\partial}{\partial r}\left(\frac{\mu}{r}\omega\right)\right) - \frac{\partial}{\partial z}\left(r^3\frac{\partial}{\partial z}\left(\frac{\mu}{r}\omega\right)\right)$$
$$+ r^2 g\frac{\partial \rho}{\partial r} + r^2\nabla\left(\frac{v_r^2 + v_z^2}{2}\right)\cdot \text{iso}\ \rho = 0. \tag{3.2}$$

Species Conservation:

$$\frac{\partial}{\partial z}\left(Y_k\frac{\partial \psi}{\partial r}\right) - \frac{\partial}{\partial r}\left(Y_k\frac{\partial \psi}{\partial z}\right) + \frac{\partial}{\partial r}\left(r\rho Y_k V_{k_r}\right) + \frac{\partial}{\partial z}\left(r\rho Y_k V_{k_z}\right)$$
$$- r W_k \dot{w}_k = 0, \quad k = 1, 2, \ldots, K. \tag{3.3}$$

Internal Energy:

$$c_p\left[\frac{\partial}{\partial z}\left(T\frac{\partial \psi}{\partial r}\right) - \frac{\partial}{\partial r}\left(T\frac{\partial \psi}{\partial z}\right)\right] - \frac{\partial}{\partial r}\left(r\lambda\frac{\partial T}{\partial r}\right) - \frac{\partial}{\partial z}\left(r\lambda\frac{\partial T}{\partial z}\right)$$
$$+ r\sum_{k=1}^{K}\left\{\rho c_{pk} Y_k\left(V_{k_r}\frac{\partial T}{\partial r} + V_{k_z}\frac{\partial T}{\partial z}\right)\right\} + r\sum_{k=1}^{K} h_k W_k \dot{w}_k = 0. \tag{3.4}$$

Other parameters appearing in this system are: the mixture viscosity μ, the acceleration of gravity g, the diffusion velocities of the k^{th} species in the mixture (V_{k_r}, V_{k_z}), the specific heat of the k^{th} species c_{pk}, the specific heat of the mixture c_p, the thermal conductivity of the mixture λ, the molecular mass of the k^{th} species W_k, and the generation/consumption rate of the k^{th} species $\dot{w}_k$.

The system is closed with the multicomponent ideal gas law,

$$\rho = \frac{p\overline{W}}{RT},$$

where $\overline{W}$ is the mixture molecular mass and p is the pressure. The Arrhenius reaction rate law for a system of J reactions gives

$$\dot{w}_k = \sum_{j=1}^{J}(\nu_{jk}^P - \nu_{jk}^R)\left[f_j - r_j\right], \tag{3.5}$$

where the forward and reverse rates are given by

$$f_j = k_j^f(T) \prod_{n=1}^{K} \left(\frac{\rho Y_n}{W_k}\right)^{\nu_{jn}^R}, \quad r_j = k_j^r(T) \prod_{n=1}^{K} \left(\frac{\rho Y_n}{W_k}\right)^{\nu_{jn}^P}, \tag{3.6}$$

where

$$k_j^f = A_j^f T^{\beta_j^f} \exp(-E_j^f/RT), \quad k_j^r = k_j^f/K_j^c, \tag{3.7}$$

and where, in turn A_j^f, the pre-exponential rate constants, β_j^f, the temperature exponents, E_j^f, the activation energies, and K_j^c, the equilibrium constants, are assumed known. The ν_{jk} are the stoichiometric coefficients of the k^{th} species in the j^{th} reaction. Superscripts P and R on the ν_{jk} denote the product and reactant sides of the stoichiometric equation, respectively. Thermodynamic and constitutive equations provide the temperature- and composition-dependent specific heats, viscosities, thermal conductivities, and diffusion velocities.

For the calculation of the $\dot{w}_k$ and the thermodynamic and constitutive properties, we use the convenient FORTRAN code packages documented in [6] and [7], respectively. Forming the effective mixture transport coefficients (μ, λ, and the V_k) by appropriate averaging of more fundamental quantities defined by the kinetic theory of gases or empirical tables is by far the most expensive part of evaluating the governing equation residuals. However, in diffusion flames, accurate diffusive transport models are essential. One dimensional studies [8] have shown, for instance, that the location of a diffusion-controlled reaction front is shifted significant distances relative to the width of the fuel-oxidizer mixing layer as the exponent of temperature variation in a very simple transport model is varied within a small neighborhood.

3.2. Discretization of the Governing System

The governing equations, along with appropriate boundary conditions, are differenced on a two-dimensional tensor product grid which (in the serial code) is generated adaptively from an initial coarse grid by subequidistribution of gradients and curvatures of the solution components. This concentrates grid points in the regions of high-activity (fronts and peaks) in the domain. Second-order differences are used throughout except for gradient boundary conditions and for the convective terms of the vorticity, energy and species equations, in which first-order upwind differences are employed. This discretization can be accommodated within the standard nine-point stencil, and, apart from the source term, insures the diagonal dominance of the Jacobian.

Ordering the solution components at each gridpoint within a lexicographical ordering of the gridpoints (radial index varying more rapidly than axial index) results in a Jacobian which has the block nine-diagonal structure indicated in Figure 1. To quantify the overall sparsity, let there be m and n gridpoints in the radial and axial coordinate directions, respectively, and r unknowns per gridpoint. The $r \times r$ blocks must be assumed fully dense to accommodate the most general kinetic mechanism and composition-dependence of the transport properties. The fraction of Jacobian elements which are nonzero is then approximately $9/(nm)$. For m and n on the order of 30, which is certainly below the minimum required for well-resolved reaction zones, this number is approximately 1%. It is therefore natural to use a block relaxation method, in which only the diagonal blocks are factored by a direct method. The relaxation may be carried out either by throwing the six blocks in the wings of the Jacobian to the right-hand side and treating the three contiguous diagonal blocks about the main diagonal implicitly (the block-line method), or of throwing all eight off-diagonal blocks to the right-hand side (the block-point method). As the outermost loop in the relaxation process used to solve the linear problems at each Newton step, the domain is swept repeatedly in the axial direction from upstream to downstream. This takes advantage of the approximately parabolic nature of all of the governing equations other than that of the streamfunction. Relaxation

is, of course, a suboptimal algorithm for solving even sparse linear systems. In order to progress to better linear solvers as we progress to denser meshes, we are presently experimenting with a block-preconditioned generalized minimum residual (GMRES) method [10] in our serial code. However, as will be noted below, the dominant cost of running the code does not necessarily come from the linear algebra, and the block relaxation schemes are attractive for their simplicity and their ease of parallelization.

The complexity of the governing equations precludes analytic expression of the elements of the Jacobian; therefore a finite-difference approximation to the Jacobian must be used. Extending the ideas of Curtis, Powell and Reid [1], as described in more detail elsewhere [14], we can form all of the nonzero elements from $(9r + 1)$ independent vector residual evaluations. Because of the large number of function evaluations required to form the Jacobian, its formation may consume nearly all of the the running time of the code. To alleviate this burden somewhat, we adopt the practice of not re-evaluating the detailed transport coefficients during Jacobian formation but lagging them at their values from the previous Newton step residual evaluation. The severity of this approximation on the nonlinear convergence rate of the algorithm in comparison to the other major Jacobian approximations (the lagging of the Jacobian itself over several Newton steps, and its approximate evaluation from finite differences) has not been studied, mainly because the control experiment (fresh evaluation of the transport terms with each Jacobian) is prohibitively expensive.

4. Parallelization Analysis

4.1. Parallel Implementation Issues

The large amount of arithmetic to be performed at each gridpoint requiring data defined only at that gridpoint suggests the potential for easy parallelization of the detailed kinetics reacting flow computation. However, there are potentially three penalties to be paid in distributing the overall relaxation-based elliptic solution algorithm over an array of independent processors: synchronization overhead, communication overhead, and degradation of convergence. These penalties are measured indirectly through the speedup and efficiency figures-of-merit of a parallel implementation. The speedup is the ratio of the uniprocessor execution time of a given algorithm to that of the multiprocessor execution of the same algorithm. The efficiency is the speedup divided by the number of processors. For many algorithms, these definitions are unnatural in the sense that one would never use the same algorithm in both uniprocessor and multiprocessor environments. (Usually *better* uniprocessor algorithms exist, so the parallel efficiency as strictly defined is inflated relative to its advantage.) We adopt measures in which the execution times are obtained from the most natural algorithm for each environment.

The synchronization penalty arises if the processors have data dependencies which cause them to cycle between computation and idle time. Even if the processors are programmed homogeneously this can happen at convergence checkpoints, for instance, or at other points where an exchange of data is required, if they have unequal amounts of work to do. The amount of work to be done in a processor is a function of the number of gridpoints in the subdomain assigned to it, the type of discrete equations to be enforced at those gridpoints, and possibly variations in the character of the solution from one gridpoint to another at which the same equations are enforced. (As instances of the latter, consider the nondeterministic arithmetic complexity of obtaining some physical parameter required in the governing equations by interpolation from a table or by solution of a small nonlinear system of equations at each grid point.) Assuming that the grid points are allocated to the processors as evenly as possible (within shape and contiguity constraints) at the beginning of each macrocycle between grid adaptations, synchronization delays are relatively unimportant in the particular problems we need to consider. The relative number of boundary gridpoints (which require somewhat less computational work than interior ones) decreases as the

mesh is refined, and though their distribution between the processors becomes more uneven, only a small number of processors are thus kept waiting. Similarly, the number of arithmetic operations at the interior gridpoints does not vary significantly despite the widely varying solution in our detailed kinetics formulation.

The communication penalty is the time spent exchanging messages between neighboring processors which depend on common data. The significance of this penalty can be great in the present context and depends on the absolute amount of data to be communicated, on the granularity of the messages into which it is embedded, on its routing between processors, on the amount of arithmetic which the algorithm must perform between exchanges, and on the communication-to-computation speed ratios of the hardware in question. Two essentially different types of exchanges are required when the serial code described in the previous section is parallelized by domain decomposition. For the nine-point second-order finite-difference stencil, evaluation of the governing equation residuals on domains decomposed into strips requires two (vector) exchanges of data per processor per iteration, and boxwise decompositions require eight such exchanges. Apart from these pairwise exchanges between processors working on adjacent subdomains, global exchanges are required for the inner products. The price of distributed memory is therefore heavy local traffic and light global traffic.

In an effort to keep the synchronization and communication overhead down, natural tradeoffs may exist which degrade the convergence rate of the algorithm by substituting readily available data for the best possible data. In the context of relaxation methods, decreasing the degree of implicitness from a Gauss-Seidel method to a Jacobi method is a natural way of decoupling an inherently sequential algorithm into independent subtasks. This decoupling may be introduced gradually as a function of the granularity of the decomposition, *e.g.*, with two processors one may employ block Jacobi between the subdomains but Gauss-Seidel within the subdomains. It can be shown theoretically [4] for the model problem of the scalar Poisson equation on the uniformly gridded unit square that the asymptotic convergence rates of the the finest granularity point or line Jacobi methods are half those of the respective Gauss-Seidel methods, so that twice as many iterations are required for an equal error reduction. Therefore, an approximate upper bound on the practical parallel efficiency for a processor-saturated Jacobi method applied to the model problem is 50%. However, the linear systems which obtain in the reacting flow context are not necessarily diagonally dominant and must, from experience, be under-relaxed. The convergence penalty incurred when under-relaxed iterations are decoupled is less than that borne by fully-relaxed iterations, as the test data of section 5 confirms. Therefore, efficiencies of greater than 50% at the processor-saturated limit are certainly not ruled out by this tradeoff.

In view of the above considerations as well as programming convenience, our parallel implementation consists of decomposing the logical tensor product computational domain into logically congruent strips or boxes of contiguous unknowns, mapping these subdomains in as neighbor-preserving a manner as possible onto network of processors, and programming the processors homogeneously. This equidistributes load, minimizes the size and distance of the messages, and simplifies the communication patterns. For stripwise decompositions, the natural candidate mappings are onto a ring of processors, a binary-reflected Gray code ring embedded in a hypercube, and a bus-connected shared memory machine. For boxwise decompositions, the natural mappings are onto a toroidal mesh of processors, a binary-reflected Gray code mesh embedded in a hypercube, and a bus-connected shared memory machine. The distributed memory mappings are chosen so that subdomains sharing common sides are assigned to processors which are directly connected. The only algorithmically significant difference between the distributed memory architectures for a given decomposition is then in their overall diameters, that is the maximum distance in message

links between any two processors, since all processors must cooperate in inner products and convergence decisions. For the ring, toroidal mesh, and hypercube, these diameters are $p/2$, $\sqrt{p}$, and $\log_2 p$, respectively. See, *e.g.*, [11, 5] for fuller descriptions of these interconnection topologies.

4.2. Parallel Complexity Analysis

A complexity analysis of a fairly involved computational procedure is most clearly approached in a modular fashion. Accordingly, we begin by constructing operation counts for the five major computational subtasks listed at the end of section 2, then combine them in proportion. A total of 15 parameters are needed to describe a model with sufficient versatility to cover the region of parameter space in which we would like to use the model now and in the near future. Better complexity models involving yet further parameters are possible, but are not necessary to obtain the basic order-of-magnitude results sought.

We begin with a fairly general model for the computer itself, since specification of the design of hardware well-suited to parallel reacting flow computations is one of our goals. A network of p homogeneous processor elements is assumed, each of which has sufficient local memory to represent the data of the associated subdomain. We define γ as the unit of scalar floating point operation time. To be precise, we count the scalar multiply-add operation as one γ. A scalar division is counted with the same weight, and an a scalar exponential is assumed to require 6γ. Any vector-processing capability of these elements is ignored. (The savings of vectorization are not negligible, even as regards the transport and source terms †, but taking advantage of this capability would require substantial recoding of our present packages, along with a more complicated complexity model.)

The processors communicate by passing messages or accessing shared memory through a global bus, depending upon the interconnection. We define α as the time overhead to initiate and route a message of any length between a pair of neighboring processors (which we neglect entirely for the shared memory machine) and β as the additional transfer time per floating point word (which is simply the reciprocal of the bus bandwidth for the shared memory machine). For simplicity, we assume that only *sending* messages (or *depositing* data in the shared memory) takes time. *Receiving* (or *reading* data) is assumed to be free. Since all sent messages are presumed received at some point, any additional reception cost which ought to be included to model a particular machine can be incorporated by simple readjustment of α and β (*e.g.*, doubling them if the two costs are comparable). We define three architecture-dependent functions of p: $C_{Ex,s}$ and $C_{Ex,c}$, the *local exchange coefficients* for processors sharing neighboring *sides* and *corners*, respectively, and C_{Re}, the *global reduction coefficient*. These coefficients multiplying α and β take into account the degraded ability of the network to exchange data as the number of processors grows. In the case of the ring, toroidal mesh, and hypercube, $C_{Ex,s}$ is one regardless of the number of processors because there are channels solely devoted to these message routes in the network. In the case of the shared-memory machine, the bandwidth to memory per processor must be divided by the number of processors; hence $C_{Ex,s}$ is p itself. In the case of the toroidal mesh and hypercube, the two distributed memory configurations for which it makes sense to introduce boxwise decompositions (and thus neighbors at the corners), $C_{Ex,c}$ is 2. This is because there are no devoted channels for these message routes and each corner-to-corner message must be routed through a mutual neighbor. (This is a conservative assumption; we ignore the fact that in some systems, the overhead of sending an k-hop message is less than k times the overhead of sending a direct, or 1-hop message.) On a bus-connected machine, messages have no preferential channels, hence $C_{Ex,c}$ is the same as $C_{Ex,s}$, namely p. The global reduction operation produces a single scalar from scalars defined on each processor, whether by addition in the case of a global inner product whose partial sums are locally accumulated, or by the logical "and" operation in the case of a convergence test. Our reduction

†Vincent Giovangigli, CNRS, Paris, personal communication (to appear)

algorithms (which, though simple, are beyond the scope of this study; see [11, 5] for a fuller discussion) are close to optimal for a reasonable "common denominator" of network software. For the ring, toroidal mesh, and hypercube, C_{Re} is twice the diameter of the network; it is $2p$ for the shared-memory machine.

The complexity model depends in addition on basic discrete problem dimensions: the resolution of the grid n (hereafter assumed the same in each direction for simplicity), the number of points in the finite difference stencil q, the number of components (or governing equations) per gridpoint r, and the number of parameters per gridpoint s. (Though q is "hard-coded" as nine in the present context, the formulae below are also valid for second-order schemes without cross-derivatives, for which $q = 5$. Generalization to higher-order schemes is not automatic, since an expanded discrete stencil would require that the processors share not only the boundary data of their associated subdomains, but successively more interior data as well.)

In analogy to the machine-dependent coefficients, there are coefficients multiplying the basic problem dimensions which depend on the details of the physics and chemistry incorporated into the governing equations. A wide variety of such physical models can be represented by simply distinguishing between three types of terms: a per-stencil-point-per-component operation count C_{CD} (whose main contribution is from the convective-diffusive terms), a per-component operation count C_{Ar} (whose main contribution is from the Arrhenius source terms), and a per-parameter operation count C_{Tr} (into which all of the transport property calculations at each grid point are lumped).

In terms of these parameters, the major algorithmic subtasks have the following complexities.

DAXPYs (strips or boxes):

$$\left[\frac{n^2}{p}r\right]\gamma.$$

The DAXPY requires no communication and parallelizes perfectly over any partitioning.

Global Norms (strips or boxes):

$$\left[\frac{n^2}{p}r\right]\gamma + [C_{Re}]\,\alpha + [C_{Re}]\,\beta.$$

The norm requires local inner products followed by a global reduction of the partial sums with addition. In the spirit of order-of-magnitude estimation, here and below, only arithmetic which is done at every gridpoint will be considered. (Thus, we do not consider the comparatively negligible cost of normalizing the norm, for instance.) However, scalar communication can be nearly as expensive as vector communication when α is large. Thus, no communication is negligible *a priori*.

Block Relaxation (strips):

$$\left[\frac{n^2}{p}(qr^2+4r)\right]\gamma + [2C_{Ex,s} + C_{Re}]\,\alpha + [2nrC_{Ex,s} + C_{Re}]\,\beta.$$

Block Relaxation (boxes):

$$\left[\frac{n^2}{p}(qr^2+4r)\right]\gamma + [4C_{Ex,s} + (q-5)C_{Ex,c} + C_{Re}]\alpha + \left[4\frac{n}{\sqrt{p}}rC_{Ex,s} + (q-5)rC_{Ex,c} + C_{Re}\right]\beta.$$

The arithmetic cost of each sweep of the block relaxation reflects the formation of a modified right-hand side (which includes a term in r^2 from each Jacobian block which is not treated implicitly),

back-substitution using pre-factored Jacobian blocks (either block tridiagonal or block diagonal, additional terms in r^2), a DAXPY update to the solution vector, and a convergence check. In the case of the stripwise decomposition, each pair of neighboring processors must exchange their bounding rows of data consisting of n gridpoints. Neighbors in boxwise decompositions are of two types: those that must exchange bounding rows of data consisting of $n/\sqrt{p}$ gridpoints, and those that must exchange corner points of data only. (The latter drop out when a 5-point stencil is employed.)

Function Evaluation (strips):

$$\left[\frac{n^2}{p}(C_{CD}qr + C_{Ar}r + C_{Tr}s)\right]\gamma + [2C_{Ex,s}]\alpha + [2n(r+s)C_{Ex,s}]\beta.$$

Function Evaluation (boxes):

$$\left[\frac{n^2}{p}(C_{CD}qr + C_{Ar}r + C_{Tr}s)\right]\gamma +$$

$$[4C_{Ex,s} + (q-5)C_{Ex,c}]\alpha + \left[4\frac{n}{\sqrt{p}}(r+s)C_{Ex,s} + (q-5)(r+s)C_{Ex,c}\right]\beta.$$

The function evaluation arithmetic cost is independent of decomposition, but the communication requirements are different for the same geometrical reasons discussed in the block relaxation step.

Jacobian Evaluation (strips):

$$\left[\frac{n^2}{p}\left((qr+1)(C_{CD}qr + C_{Ar}r) + 2qr^2 + 2r + \frac{7}{3}r^3\right)\right]\gamma +$$

$$[(2(qr+1) + (q-3))C_{Ex,s}]\alpha + [(2(qr+1)n(r+s) + (q-3)nr^2)C_{Ex,s}]\beta.$$

Jacobian Evaluation (boxes):

$$\left[\frac{n^2}{p}\left((qr+1)(C_{CD}qr + C_{Ar}r) + 2qr^2 + 2r + \frac{1}{3}r^3\right)\right]\gamma +$$

$$[4(qr+2)C_{Ex,s} + (q-5)(qr+2)C_{Ex,c}]\alpha +$$

$$\left[4\frac{n}{\sqrt{p}}((qr+1)(r+s) + r^2)C_{Ex,s} + (q-5)((qr+1)(r+s) + r^2)C_{Ex,c}\right]\beta.$$

The Jacobian evaluation cost includes $(qr+1)$ independent function evaluations, in which the dependence on the detailed transport through C_{Tr} is neglected, along with some finite difference arithmetic performed on the function evaluations to generate the matrix elements, and some preprocessing which is amortized over all of the block relaxation sweeps utilizing the Jacobian. The stripwise method preprocessing cost of $7r^3/3$ reflects an LU-block factorization of the diagonal block and the solution of r-dimensional linear systems involving the columns of the two neighboring off-diagonal blocks. In the boxwise method (the most fine-grained limit of which is block-point relaxation) the preprocessing cost is just $r^3/3$ per gridpoint for factorization of the diagonal block. The communication terms include both the sharing of subdomain boundary data needed for the function evaluation and the shipping of Jacobian blocks between neighboring processors. The latter is necessary in order to exploit the Curtis-Powell-Reid (CPR) technique refered to in section 3. In the CPR method, it is easier to compute, *e.g.*, $\frac{\partial F_{i,j+1}}{\partial \phi_{i,j}}$ at gridpoint (i,j) than it is to compute $\frac{\partial F_{i,j}}{\partial \phi_{i,j-1}}$, where $F_{i,j}$ represents any of the governing equation residuals at the point and $\phi_{i,j}$ any of

the unknowns. However, it is $\frac{\partial F_{i,j}}{\partial \phi_{i,j-1}}$ which is actually needed locally for the linear algebra. If the point (i,j) in question lies at the low-j end of a subdomain then this quantity is formed at the high-j end of the neighboring subdomain as $\frac{\partial F_{i,j+1}}{\partial \phi_{i,j}}$, and so forth, then the blocks are swapped. This approach is justified since the block-swapping requires no arithmetic and it does not contribute to the leading communication term in α when r is large. The extra function evaluations which would otherwise be required would contribute to the leading terms in each of the brackets multiplying γ, α, and β.

If the linear systems involving the Jacobian are solved to a level of convergence which is independent of the granularity of the decomposition, then the number of hybrid Newton steps is independent of the type of decomposition and the number of processors. † We may therefore use the hybrid Newton step as the largest aggregate in the complexity model, and work out the complexity on a per-call-to-**NEWTON** basis. Neglecting the overhead associated with possible problem-dependent damping and selection of time steps, we can state that each hybrid Newton step consists of one DAXPY, two norms (one of the solution vector update for determining convergence, and one of the steady-state residual norm for additional monitoring), one function evaluation, L relaxation sweeps, and $1/K$ Jacobian evaluations. The last two parameters of the model are therefore the average number of relaxation sweeps per Newton step, L, and the average number of Newton steps between Jacobian evaluations, K. For systems as complicated as (3.1)-(3.7), no theory exists on either L or K. We must therefore obtain L and K from problem-specific serial experiments, and be content to extrapolate them into other regions.

To illustrate the usefulness of the complexity analysis while keeping the size of this paper within reason, it is convenient to focus on individual points in physical parameter space and to study the parallelizability as a function of n, p, and the dimensionless ratios $a \equiv \alpha/\gamma$ and $b \equiv \beta/\gamma$ alone. For this purpose, we adopt the specific 16-species, 46-reaction mechanism of [9] for the oxidation of methane. We then have for the basic problem dimensions: $q = 9$, $r = 19$ (with unknowns $\psi, \omega, T, Y_1, Y_2, \ldots, Y_{16}$), and $s = 20$ (with parameters $\rho, c_p, \mu, \lambda, D_1, D_2, \ldots, D_{16}$). The operation count coefficients are approximately: $C_{CD} \approx 33$, $C_{Ar} \approx 270$, and $C_{Tr} \approx 2000$. The coefficient C_{Ar} is obtained directly from counting the operations required to evaluate the species and energy source terms, taking the sparsity of the sums and products of (3.5)-(3.7) into account, and dividing by r. The other two coefficients are too difficult to estimate directly from the code itself, so they are fitted to C_{Ar} for a particular q, r, and s by the simple formula:

$$\frac{C_{Ar}r}{T_{Ar}} = \frac{C_{CD}qr}{T_{CD}} = \frac{C_{Tr}s}{T_{Tr}},$$

where T_{Ar}, T_{CD}, and T_{Tr} are actual serial CPU timings of the respective portions of the code which performs the function evaluation. $L \approx 200$ and $K \approx 5$ are also available from actual runs of the serial code on a grid of approximately 1200 nonuniformly-spaced points.

4.3. Parallel Performance Analysis

With the preceding formulae and assigned values, we can estimate the execution time T of the parallelized solver (normalized to the floating point speed of the processors) through a relation of the form:

$$\frac{T}{\gamma} = f_\gamma(n,p) + a f_\alpha(n,p) + b f_\beta(n,p). \tag{4.1}$$

Under the additional assumptions that $a = 100$ and that $b = 1$ the leading-order communication terms of can easily be extracted, leading to the entries under f_γ, af_α, and bf_β in Table 1. Minimizing

†This condition may not be satisfied for relaxation methods in which convergence is based solely upon the size of the update, since the deteriorating condition number of the iteration matrix with finer granularity may allow a larger residual at convergence. However, this has a relatively small effect on the total number of Newton steps.

T/γ as a function of p leads to the optimal execution time, T_{opt}, for a given n, a, and b. The number of processors which minimizes T/γ for each of the six mappings described above is given in the fourth column of Table 1 in orders of n. The fifth column contains the actual optimum number of processors for the case $n = 64$ when all the terms of the complexity model are taken into account, and the last column shows the fraction of the total execution time which is devoted to communication at this optimum. (The value $n = 64$ is selected because one-dimensional studies of methane-air diffusion flames [8] suggest that this should be sufficient resolution in either direction for approximate grid-independence of the discrete solution.)

Mapping	$f_\gamma/10^6$	$af_\alpha/10^4$	$bf_\beta/10^4$	$p_{opt,\partial}$	p_{opt}	comm. frac.
strips/cube	$1.1\ n^2p^{-1}$	$4.0\ \log_2 p$	$1.1\ n$	$O(n^2)$	64	0.014
strips/ring	$1.1\ n^2p^{-1}$	$2.0\ p$	$1.1\ n$	$O(n)$	64	0.028
strips/bus	$1.1\ n^2p^{-1}$	-	$1.1\ np$	$O(n^{1/2})$	64	0.39
boxes/cube	$1.1\ n^2p^{-1}$	$4.0\ \log_2 p$	$2.1\ np^{-1/2}$	$O(n^2)$	4096	0.43
boxes/mesh	$1.1\ n^2p^{-1}$	$4.0\ p^{1/2}$	$2.1\ np^{-1/2}$	$O(n^{4/3})$	3686	0.70
boxes/bus	$1.1\ n^2p^{-1}$	-	$2.1\ np^{1/2}$	$O(n^{2/3})$	206	0.52

Table 1: Optimal processor estimates for six subdomain-to-architecture mappings, and estimates of the fraction of the total execution time devoted to communication at the optimal p, assuming $n = 64$, $a = 100$, and $b = 1$.

In the case of stripwise decomposition on a hypercube, the optimal order of n^2 is too optimistic, since it the algorithm can only employ as many processors as there are strips. (By construction, $1 \le p \le n$ for the stripwise decompositions and $1 \le p \le n^2$ for the boxwise decompositions.) The communication fraction remains very small (less than 2%) when this decomposition is processor-saturated at $p = n = 64$. From a user perspective, this order of speedup reduces an hour of serial execution time to approximately one minute.

On a ring, the communication fraction for processor-saturated case is only slightly worse (about 3%). The only difference in complexity between the two cases is a larger global reduction time due to the fact that the ring interconnect is less rich. However, global reductions form a small proportion of the total running time in this region of parameter space.

In the case of stripwise decomposition on a bus-connected shared memory machine the bus becomes choked with local exchanges as p increases leading to an optimal order of $n^{1/2}$. However, this is too pessimistic for reasonable values of n and the chosen machine parameters since the constant in front is large and the optimum actually lies on the boundary. Although the execution time continues to drop as processors are added to the choked bus, the efficiency drops considerably. In the processor-saturated case, nearly 40% of the time is devoted to communication.

In the case of boxwise decomposition on a hypercube, the optimal order estimate of n^2 (which is derived from the balance between f_γ and f_α alone) is justified by the full model, in that $64^2 = 4096$ processors can usefully be employed. However, at this processor-saturated limit the speedup is limited since over 40% of the time is devoted to communication. (For $a = 100$ and $b = 1$, the communication is dominated by the global reduction operation.) Nevertheless, from a user perspective, a processor array of this size could reduce about 40 minutes of serial execution time to approximately one second.

The other architectures do not perform as well for the boxwise decomposition. The difference between the $2\log_2 p$ global reduction coefficient for the hypercube and the corresponding $2\sqrt{p}$ for the mesh is significant for p on the order of 2^{12}, and the optimal order falls to $n^{4/3}$. This is somewhat

pessimistic for the problem under consideration because the constant in front is greater than one, and the execution time is actually minimized at $p = 3686$. However, communication costs 70% of the execution time at this point.

The bus-connected shared memory machine becomes hopelessly choked with a mere 206 processors under this decomposition. Further increases in the number of processors actually *increase* the execution time. At this point, over 50% of the execution time is spent swapping data. The optimal order of $n^{2/3}$ is not much better than in the stripwise decomposition case.

The optimum number of processors has been defined above as the number which minimizes the execution time. However, the "optimum" might actually be smaller if the marginal efficiency of the processors is small and a cost-weighted objective function is used. Because of this important consideration, we conclude this section with the presentation of surface plots of efficiency over a range of of n and p for the complexity model defined above. (For plotting clarity, the efficiency is set to zero in the inaccessible ranges of $p > n$ for stripwise decompositions or $p > n^2$ for boxwise decompositions.) Results are given for all six mappings for the machine parameters $a = 100$ and $b = 1$, and for the two hypercube mappings (which are the most promising) at $a = 1000$ and $b = 1$, in order to check the sensitivity of their performance to the message initiation overhead. Figure 2 contains the strip-based mappings, and Figure 3 the box-based mappings.

It is apparent from Figure 2(d) that the hypercube/strip mapping can bury the cost of initiating messages over a rather slow network, whereas the performance of hypercube/box mapping shown in Figure 3(d) has deteriorated to possibly uneconomical levels. Further calculations (not plotted) show that the hypercube/strip mapping maintains greater than 70% efficiency even at $a = 10,000$.

In all the calculations of this section, L was set to the constant value 200 obtained from serial runs with the block-line algorithm. This crude assumption potentially endangers the results, making them appear too optimistic at large n and p, since L appears linearly in the leading terms of the communication. (It does not appear in the leading terms of the arithmetic, since the Jacobian and function evaluations dominate.) By analogy with a scalar Poisson equation, L might be expected to increase like n^2 at a fixed p, by as much as a factor of 2 at fixed n over the full range of p for the stripwise decomposition, and by as much as a factor of 4 over the full range of p in the boxwise decomposition case (see, *e.g.*, [4]). However, experience indicates that these scalar Poisson-based iteration estimates are pessimistic for the Jacobians of two-dimensional strongly convective systems of reacting flow equations. Still, to be conservative, these optimal p estimates should be regarded as valid only in the neighborhood of the parameter space in which they were derived. In particular, they are *not* uniformly valid in n. In order to model performance conservatively over a wide region of parameter space with the convenient assumption of a constant L, the worst-case L for the region (based on its extreme point in n and p) should be used. This number is not yet experimentally available to us for $n = 64$, the extreme resolution in Figures 2 and 3.

5. Parallel Experiments for Model Problems

The solution algorithm described in section 2 has been implemented on the 32-processor Intel iPSC-5 (with the expanded local memory configuration) and some results are presented in this section for a few simple choices of $F(\phi)$. These results serve three main purposes: *(i)* they validate the parallel code itself, *(ii)* they provide a test for the complexity theory of section 4 in the case of constant L, and *(iii)* they illustrate that the dependence of the overall performance of the code on variations in L with n or p is weaker than one would expect from the model problem, under various conditions which typify our reacting flow calculations. These conditions are: under-relaxation of the linear solver, strong (upwinded) convection, and large r with strong coupling of the r unknowns at each point.

The model Poisson problem is

$$-\nabla^2 u = f$$

on the unit square with homogeneous Dirichlet boundary conditions, where $f(x,y)$ is chosen so that the exact solution is $u(x,y) = \sin(\pi x)\sin(\pi y)$. The initial iterate is $u = 0$. The stripwise decomposition is applied to a uniform 32×32 grid, using 1, 2, 4, 8, 16, and 32 processors. Because the problem is linear, convergence is obtained in one Newton step. An absolute convergence tolerance of 10^{-6} on the normalized 2-norm of the update is applied to the linear iteration, in which a relaxation factor of unity is employed.

The results appear in Table 2. The first three rows display convergence data: the number of iterations required for a given granularity of the decomposition, this number normalized to the uniprocessor case, and the relative reduction of the 2-norm of the residual at "convergence". One observes that the number of iterations required for the processor-saturated $p = 32$ (block-line Jacobi) case is nearly double that of the $p = 1$ (block-line Gauss-Seidel) case. It is less than the theoretical asymptotic factor of two since this factor is based on equal error reduction, and such a condition is not guaranteed with an update-based convergence criterion. The next three rows give overall performance measures: the execution time in seconds (which includes a single Jacobian formation at the outset), the relative speedup obtained from each doubling of the number of processors (ideally 2 each time), and the efficiency ($E = T_1/(pT_p)$). Though the speedup is respectable (from a little over half-an-hour to three minutes), the efficiency is fairly dismal (32%) at the processor-saturated limit due to two factors: the large communication-to-arithmetic ratio of the $r = 1$ problem and the increase of L with p. (The average number of relaxation sweeps per Newton step L is, of course, the same as the tabulated I for this linear problem.) The latter dependence can be removed by considering the execution time per iteration. Performance data based thereon, analogous to the middle three rows, appears in the last three rows of Table 2. By this measure, the processor-saturated efficiency is 58%, and its departure from unity is due exclusively to communication-related factors.

	$p=1$	2	4	8	16	32
I_p	451	477	501	550	646	832
I_p/I_1	1.00	1.06	1.11	1.22	1.43	1.85
$\|\|r_{I_p}\|\|/\|\|r_0\|\|$	9.5(-4)	1.0(-3)	1.2(-3)	1.4(-3)	1.6(-3)	2.0(-3)
T_p (sec.)	1840.	974.4	539.3	314.1	214.0	182.5
T_{p-1}/T_p	-	1.89	1.81	1.71	1.47	1.17
E	1.00	0.94	0.85	0.73	0.54	0.32
t_p (sec.)	4.08	2.04	1.08	0.57	0.33	0.22
t_{p-1}/t_p	-	2.00	1.89	1.89	1.73	1.50
e	1.00	1.00	0.94	0.89	0.77	0.58

Table 2: Intel Hypercube data for the model linear problem, $-\nabla^2 u = f$ in the unit square, with $\omega = 1.0$, for different numbers of processors, p. I_p is the number of iterations required for convergence, $||r_{I_p}||/||r_0||$ is the ratio of final to initial residuals, T_p is the execution time, E is the overall efficiency of the p-processor algorithm, t_p is the per iteration CPU time, and e is the per iteration efficiency of the p-processor algorithm.

The last row of the table can be directly compared with a theoretical efficiency surface plot for the hypercube/strip mapping of the model problem, which appears in Figure 4(a). This plot was generated using the complexity model of the previous section with $q = 9$ †, $r = 1$, $s = 1$,

† Though the model problem could be differenced with a five-point formula, the nine-point Jacobian is presently hard-coded

$C_{CD} = 1.3$, $C_{Ar} = 29$ ‡, $C_{Tr} = 0$, $L = 451$, and $K = 1$, along with $\alpha = 1000$ μsec, $\beta = 8$ μsec, and $\gamma = 40$ μsec ¶, or $a = 25$, $b = 0.2$. An interesting trend which is barely discernable in Figure 2 (b) and (c), but readily apparent in Figure 4(a) is the increase in parallel efficiency with n along the processor-saturated diagonal ($p = n$). The leading-order arithmetic terms grow faster with n than the leading-order communication terms for the hypercube/strip mapping, so that parallel efficiency improves with grid refinement. We emphasize that this is an artifact of a constant L.

Figure 4(b) contains two sections of the surface in (a), showing the theoretical efficiency at two fixed values of n as p varies from 1 to n. Plotted as circles on (b) are the data from the last row of Table 2 and analogous results on an $n = 16$ problem (not separately tabulated). The agreement of the crude complexity model with experiments is better than one has a right to expect. We emphasize that we would never use a relaxation algorithm to solve this simple problem, even in parallel, since many efficient alternative solution techniques exist and have been parallelized (FFTs, multigrid, conjugate-gradient-based domain decomposition, etc.). However, it is a useful test problem since reacting flow problems often possess regions in which diffusion dominates or co-dominates. In addition, the streamfunction equation is simply a variable-coefficient Poisson equation.

Table 3 illustrates the decline in the iteration penalty associated with loss of implicitness when the updates are under-relaxed. The problem is the same as the previous one, except that $\omega = 0.5$, which is the value we are typically forced to use in solving systems based on the reacting flow Jacobian. Though the total number of iterations is higher at all granularities, it grows more slowly, reaching a maximum of 1.28 times the uniprocessor total, rather than 1.85 for the fully-relaxed calculation in Table 2. The overall parallel efficiency is accordingly higher. The per-iteration efficiency data is identical (to within the precision of the table) to that of Table 2, since L is normalized out, and therefore is not repeated here.

	$p = 1$	2	4	8	16	32
I_p	1192	1214	1235	1278	1362	1529
I_p/I_1	1.00	1.02	1.04	1.07	1.14	1.28
$\|\|r_{I_p}\|\|/\|\|r_0\|\|$	2.9(-3)	3.1(-3)	3.2(-3)	3.3(-3)	3.5(-3)	4.0(-3)
T_p (sec.)	4870.	2485.	1334.	731.0	453.5	335.2
T_{p-1}/T_p	-	1.95	1.86	1.82	1.61	1.35
E	1.00	0.98	0.91	0.83	0.67	0.45

Table 3: Intel Hypercube data for the model linear problem, $-\nabla^2 u = f$ in the unit square, with $\omega = 0.5$, for different numbers of processors, p. I_p is the number of iterations required for convergence, $||r_{I_p}||/||r_0||$ is the ratio of final to initial residuals, T_p is the execution time, and E is the overall efficiency of the p-processor algorithm.

Table 4 illustrates the decline in the iteration penalty which is normally associated with elliptic character when the system is, in fact, approximately parabolic and the relaxation sweeps are done in the proper direction. The problem (from [2]) is

$$-\nabla^2 u + cu_y = 0$$

into the solver.

‡The C_{CD} and C_{Ar} operation count coefficients are based on detailed timing tests on a single iPSC node. The C_{Ar} term includes the cost of evaluating the sine functions in f.

¶Double precision is performed in software.

on the unit square with the inhomogeneous Dirichlet boundary conditions $u(0,y) = u(1,y) = 1$, $u(x,0) = 1 + \sin(\pi x)$, $u(x,1) = 1 + 2\sin(\pi x)$, whose solution possesses a downstream boundary layer whose resolution requirements we ignore. The stripwise decomposition is applied to a uniform 32×32 grid, using 1, 2, 4, 8, 16, and 32 processors. The initial iterate is $u = 0$. Since this is simply another scalar linear problem, we present only the iteration counts in Table 4.

Values of the dimensionless Reynolds/Peclet number c in the jet configuration which motivates this study are typically in the range 10^2 to 10^3. Therefore we consider both magnitudes below. (The first row of Table 2 gives the $c = 0$ limit.) In order to indicate the effect of n upon L, we also consider two resolutions, $n = 16$ and $n = 32$, at each Reynolds/Peclet number.

	$p=1$	2	4	8	16	32
I_p $(c = 10^2, n = 16)$	19	23	27	34	49	-
I_p $(c = 10^2, n = 32)$	42	45	50	59	75	107
I_p $(c = 10^3, n = 16)$	8	11	13	18	27	-
I_p $(c = 10^3, n = 32)$	12	14	17	22	32	51

Table 4: Intel Hypercube data for the strongly convective linear problem, $-\nabla^2 u + cu_y = 0$ in the unit square, for different numbers of processors, p. I_p is the number of iterations required for convergence with c and n as indicated.

There are three major trends in Table 4. The first is that the number of iterations required in this range of c is roughly an order of magnitude less than for $c = 0$, for the same n and p. In nonlinear problems especially, this translates into less communication per Jacobian and function evaluation, and thus into higher efficiencies. One also observes that doubling the resolution of the grid generally less than doubles the number of iterations for a given c and p. In contrast, for the model Poisson problem, doubling the resolution theoretically quadruples the number of iterations. In the other limit, $c \to \infty$, of course, the number of iterations is independent of the resolution altogether, namely p itself. Thirdly, the number of iterations may more than double for a given c and n as p increases over its full range, the $c \to \infty$ limit being a case in point. In this respect, the Poisson problem has a milder dependence of L upon p. However, in that problem, L is much larger to begin with.

Of course, the effect of convection on the reacting flow operator is far more complex than can be suggested here, since there is one equation in the governing system (3.1) in which convection plays no role at all. In addition, convection is not generally a co-dominant term within the reaction zone, so that it is insufficient simply to march through this zone a small number of times.

Finally, to verify the handling of nonlinearities and to illustrate improved overall efficiency with large r, we consider the problem

$$-\nabla^2 u_k + c \exp[\sum_{l=1}^{r} a_{kl} u_l] = f_k, k = 1, \ldots, r$$

on the unit square with homogeneous Dirichlet boundary conditions, where $a_{kl} = +1$ for $l \leq k$ and $a_{kl} = -1$ for $l > k$, and where the f_k are the same as in the model problem, so that all of the u_k reduce to the solution thereto when $c = 0$. Here we take $c = 4(n-1)^2$ so that the nonlinear term is co-dominant. However, no time-stepping is employed, and no damping is triggered by this particular problem for the initial iterate $u = 0$.

The stripwise decomposition is applied to a uniform 32×32 grid, using 16 and 32 processors, for 1, 2, 3, and 4 unknowns per gridpoint. In order to keep execution times under one hour, smaller

numbers of processors (and larger numbers of unknowns) were not used; therefore direct efficiency calculations are not possible. The results appear in Table 5. Notice that for a given n and p, L levels off as r increases and the intrapoint couplings (which are handled directly) become more important relative to the interpoint couplings (which are not). The relative speedups are improving with r at this processor-saturated end of the scale due to the increasing arithmetic-to-communication ratio It is primarily this effect which grants the high efficiencies of the parallelized reacting flow solver.

	$r=1$	2	3	4
N	15*	17*	21	24
J	5	6	6	6
F	20*	23*	27	30
I_{16}	927	1531	2042	2332
I_{32}	1160	1926	2561	2924
L_{16}	61.8	90.1	97.2	97.2
L_{32}	72.5	120.4	121.9	121.8
T_{16} (sec.)	318.2	957.4	2109.	3576.
T_{32} (sec.)	262.7	704.3	1470.	2406.
T_{16}/T_{32}	1.21	1.36	1.43	1.49
t_{16} (sec.)	0.34	0.62	1.03	1.53
t_{32} (sec.)	0.23	0.37	0.57	0.82
t_{16}/t_{32}	1.55	1.68	1.81	1.87

Table 5: Intel Hypercube data for the model coupled nonlinear problem, $-\nabla^2 u_k + c\exp(\sum_{l=1}^{r} a_{kl}u_l) = f_k$, for $k = 1,\ldots,r$ in the unit square, with $\omega = 1.0$, for four different numbers of components, $r = 1,\ldots,4$, and two different numbers of processors, $p = 16, 32$. N is the number of Newton iterations, J the number of Jacobian evaluations, and F the number of function evaluations. I_p is the iteration count for the linear relaxation sweeps, totaled over all N Newton steps, and L_p is the average per Newton step. T_p is the CPU time in seconds, and t_p is the per iteration CPU time in seconds. [* The asterisked figures are for the 16-processor case. Due to the slight difference in size of the terminal residuals of the linear iterations within each Newton step between the two different processor configurations, the number of Newton steps and function evaluations varied slightly. In the 32-processor case N and F were each one greater for $r = 1$, and one less for $r = 2$. The non-asterisked figures for N, J, and F are the same for both processor configurations.]

6. Conclusions

We conclude with a discussion of an instance of the "inverse problem" of parallel computing, namely, "Given problem X and algorithm Y, what kind of parallel computer should it be executed on?" For a generic two-dimensional nonlinear elliptic system characterized by very expensive function evaluations, a Newton-like method with a relaxation-based solver of either block-line or

block-point type forms a practical algorithm. The block-line form of the algorithm parallelizes well on a ring or hypercube network of processors, and efficiently employs any number of processors less than or equal to n, the resolution in the coordinate direction transverse to the lines, while tolerating a relatively high message initiation to floating point speed ratio. Shared memory machines are also useful for this problem, particularly in the coarse-grained limit. This implies that several of the currently available parallel clusters containing small numbers of powerful nodes will be of great interest to reacting flow modelers.

To achieve acceptable running times, however, modelers ultimately will not be content with speedups of $O(n)$. The block-point form of the algorithm also lends itself to parallel implementation, and extends the number of processors that can effectively be put to use beyond $O(n)$ into the $O(n^2)$ range without any necessity of decoupling the strongly interrelated unknowns defined at a single gridpoint. The block-point version places more restrictions on the type of interconnect, however. With realistic message overheads a hypercube is required for high performance. No hypercubes of the power and dimension required for this algorithm exist today; however, their development in the near future is both likely and to be welcomed.

We have considered only two-dimensional detailed kinetics models. Parallelism is not a critical issue in zero- or one-dimensional detailed kinetics models, and no projections are offered on three-dimensional models. Though the analysis of this paper could easily be extended to three dimensions, practical three-dimensional models will almost certainly require better methods for the discretization and the linear algebra. (Particle-based methods with their own very different parallelization considerations may ultimately replace continuum-based methods in both two and three dimensions.)

To borrow the now famous phrase of C. Moler, detailed kinetics reacting flow modeling is an example of an "embarrassingly parallel" computation. The Newton/time-stepping hybrid algorithm will run well on fine-grained versions of technologically feasible machines. With reasonable spatial resolution, on the order of 2^{10} or more processors may usefully be employed, although it is likely that better algorithms will be developed before the present one is implemented on hardware of this scale. The basic property of reacting flows leading to the ease of parallelization – the large amount of zero-space-dimensional arithmetic required in the evaluation of transport and reaction rate terms – will only be accentuated following present trends towards heavier fuels and the development of progressively more complicated mechanisms.

7. Acknowledgements

The authors would like to thank Dr. Robert G. Voigt of ICASE and Professor Martin H. Schultz of the Yale Research Center for Scientific Computation for their encouragement and support of this investigation. The serial modeling effort (to be reported elsewhere) is supported by a grant for computer time at the NSF Cornell Production Supercomputing Facility. The authors also acknowledge their indebtedness to Professor William D. Gropp of Yale University for paving their way into parallel computing with many original program development and analysis tools.

8. Appendix

The Appendix contains procedural level language descriptions of the procedures **SOLVER** and **NEWTON** described in section 2.

Procedure SOLVER(**F**, $\tilde{x}$, n_{time}, Δt_1, $flag_{SOLVER}$ **)**
$x_0 \leftarrow \tilde{x}$
$flag_{JACOBIAN} \leftarrow$ "refresh"
If ($n_{time} > 0$) **Then**

```
    For k = 1 Step 1 Until n_time Do
        Repeat
            x ← x_{k-1}
            Call NEWTON( F, x, n_time, x_{k-1}, Δt_k, flag_NEWTON, flag_JACOBIAN )
            If (flag_NEWTON = "converged") Then
                x_k ← x
                Break_Repeat
            Else
                Δt_k ← Δt_k/2
                If (Δt_k < Δt_min) Then
                    flag_SOLVER ← "damped to death in transient mode"
                    Return
                End_If
                x ← x_{k-1}
                flag_JACOBIAN ← "refresh"
            End_If
        End_Repeat
        compute next time step, Δt_{k+1}
        If (Δt_{k+1} > Δt_max) Then
            Break_For
        End_If
    End_For
    flag_JACOBIAN ← "refresh"
    n_time ← 0
End_If
Call NEWTON( F, x, n_time, x_{k-1}, Δt_k, flag_NEWTON, flag_JACOBIAN )
If ( flag_NEWTON ="converged") Then
    x̃ ← x
    flag_SOLVER ← "converged to steady solution"
Else
    flag_SOLVER ← "damped to death in steady-state mode"
End_If
Return
End_Procedure
```

```
Procedure NEWTON ( F, x̃, n_time, x_{k-1}, Δt_k, flag_NEWTON, flag_JACOBIAN )
x ← x̃
m ← 1
For k = 1 Step 1 Until n_NEWTON Do
    If (flag_JACOBIAN = "refresh") Then
        compute new Jacobian, J
        flag_JACOBIAN ← "recycle"
    End_If
    If (m = 1) Then
    Call F( x, n_time, x_{k-1}, Δt_k, F )
        Δx ← −J^{-1}F
        compute initial damping parameter, λ
        ξ ← ||Δx||
```

```
        x_save ← x
        Repeat
            x ← x + λΔx
            Δx_look ← −J^{-1}F(x)
            ξ_look ← ||Δx_look||
            If (ξ_look < ξ_converge) Then
                flag_NEWTON ← "converged"
                x̃ ← x
                Return
            Else_If (ξ_look < α_2 ξ And λ = 1) Then
                m ← 2
                ξ_save ← ξ
                Break_Repeat
            Else_If (ξ_look < ξ) Then
                flag_JACOBIAN ← "refresh"
                Break_Repeat
            Else
                λ ← λ/2
                If (λ < λ_min) Then
                    flag_NEWTON ← "damped to death"
                    Return
                End_If
                x ← x_save
            End_If
        End_Repeat
    Else
        x ← x + Δx_look
        Call F( x, n_time, x_{k-1}, Δt_k, F )
        Δx_look ← −J^{-1}F
        ξ_look ← ||Δx_look||
        If (ξ_look < ξ_converge) Then
            flag_NEWTON ← "converged"
            x̃ ← x
            Return
        Else_If (ξ_look < α_m ξ_save And m < n_modified) Then
            m ← m + 1
        Else
            m ← 1
            flag_JACOBIAN ← "refresh"
        End_If
    End_If
End_For
flag_NEWTON ← "failed to converge"
Return
End_Procedure
```

References

[1] A. R. Curtis, M. J. Powell & J. K. Reid, *On the Estimation of Sparse Jacobian Matrices,* J. Inst. Math. Appl., 13 (1974), pp. 117–119.

[2] E. C. Gartland, Approximation of a Nonsymmetric, Singularly Perturbed Problem, G. Birkhoff & A. Schoenstadt ed., *Elliptic Problem Solvers II,* Academic Press, New York, 1984, pp. 545–555.

[3] A. D. Gosman, W. M. Pun, A. K. Runchal, D. B. Spalding & M. Wolfshtein, *Heat and Mass Transfer in Recirculating Flows,* Academic Press, New York, 1969.

[4] L. A. Hageman & D. M. Young, *Applied Iterative Methods,* Academic Press, New York, 1981.

[5] S. L. Johnsson, *Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures,* Technical Report 361, Dept. of Computer Science, Yale University, September 1985.

[6] R. J. Kee, J. A. Miller & T. H. Jefferson, *CHEMKIN: A General-Purpose, Transportable, Fortran Chemical Kinetics Code Package,* Technical Report SAND80-8003, Sandia National Laboratories, March 1980.

[7] R. J. Kee, J. Warnatz & J. A. Miller, *A FORTRAN Computer Code Package for the Evaluation of Gas-Phase Viscosities, Conductivities, and Diffusion Coefficients,* Technical Report SAND83-8209, Sandia National Laboratories, March 1983.

[8] D. E. Keyes & M. D. Smooke, *Flame Sheet Starting Estimates for Counterflow Diffusion Flame Problems,* Technical Report ME-102-86, Dept. of Mechanical Engineering, Yale University, March 1986. (to appear in J. Comp. Phys.).

[9] J. A. Miller, R. J. Kee, M. D. Smooke & J. F. Grcar, *The Computation of the Structure and Extinction of a Methane-Air Stagnation Point Diffusion Flame,* Technical Report WSS/CI 84-20, The Combustion Institute, 1984. (presented at the 1984 Spring Meeting of the Western States Section of the Combustion Institute, University of Colorado, Boulder).

[10] Y. Saad and M. H. Schultz, *GMRES: A Generalized Minimum Residual Algorithm for Solving Nonsymmetric Linear Systems,* Technical Report YALEU/DCS/RR–254, Dept. of Computer Science, Yale University, August 1983.

[11] ————, *Data Communication in Parallel Architectures,* Technical Report YALEU/DCS/RR–461, Dept. of Computer Science, Yale University, March 1986.

[12] M. D. Smooke, *Solution of Burner-Stabilized Pre-Mixed Laminar Flames by Boundary Value Methods,* J. Comp. Phys., 48 (1982), pp. 72–105.

[13] ————, *An Error Estimate for the Modified Newton Method with Applications to the Solution of Nonlinear Two-Point Boundary Value Problems,* J. Opt. Theory and Appl., 39 (1983), pp. 489–511.

[14] M. D. Smooke, R. E. Mitchell & J. F. Grcar, Numerical Solution of a Confined Laminar Diffusion Flame, G. Birkhoff & A. Schoenstadt ed., *Elliptic Problem Solvers II,* Academic Press, New York, 1984, pp. 557–568.

[15] M. D. Smooke, J. A. Miller & R. J. Kee, *Solution of Premixed and Counterflow Diffusion Flame Problems by Adaptive Boundary Value Methods,* Num. Meth. for Two-Point Boundary Value Problems, 5 (1985), pp. 303–317.

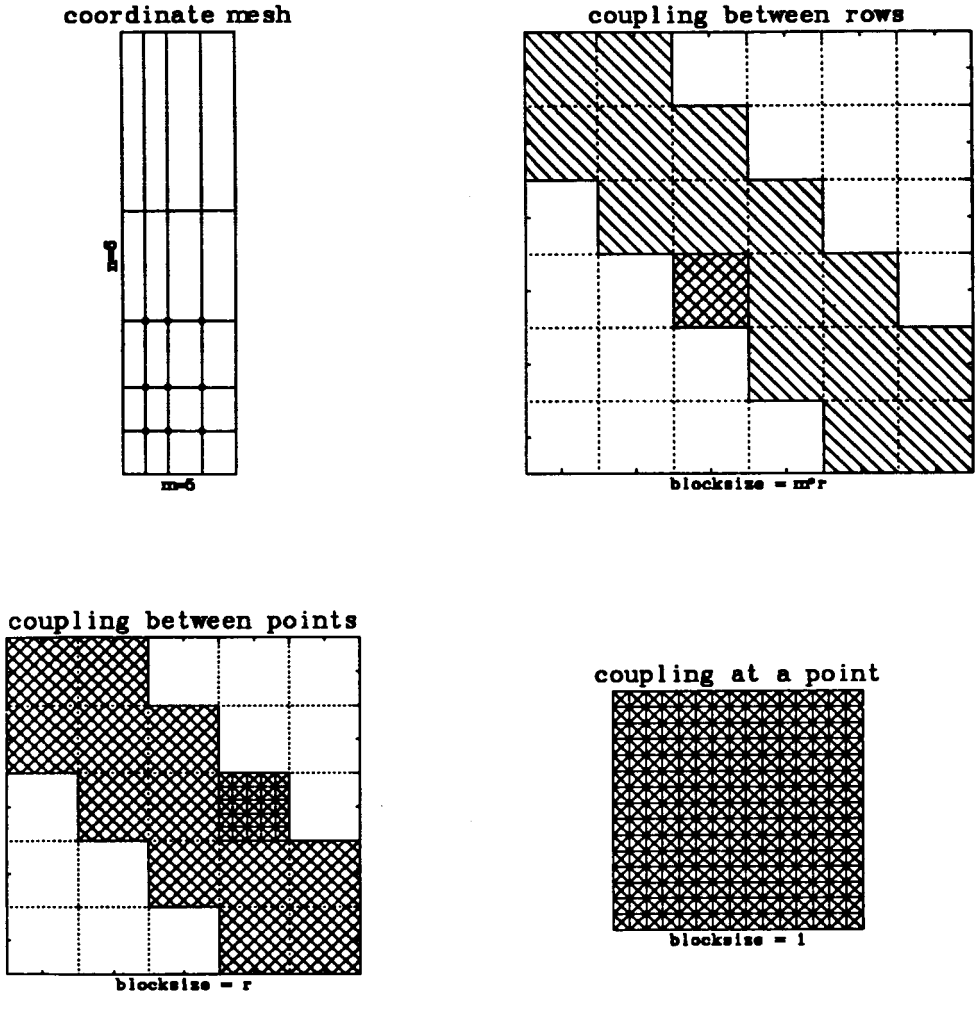

Figure 1: A schematic of the Jacobian matrix of the 9-point finite-difference operator showing its sparsity structure. (a) upper left: a two-dimensional 5×6 grid with the stencil corresponding to $(i, j) = (3, 3)$ highlighted. (b) upper right: gross sparsity structure of the Jacobian, showing the coupling between rows. (c) lower left: an enlargement of the cross-hatched block of (b), showing the coupling between points within a row. (d) lower right: an enlargement of the double-cross-hatched block of (c), showing the coupling between degrees of freedom within a point.

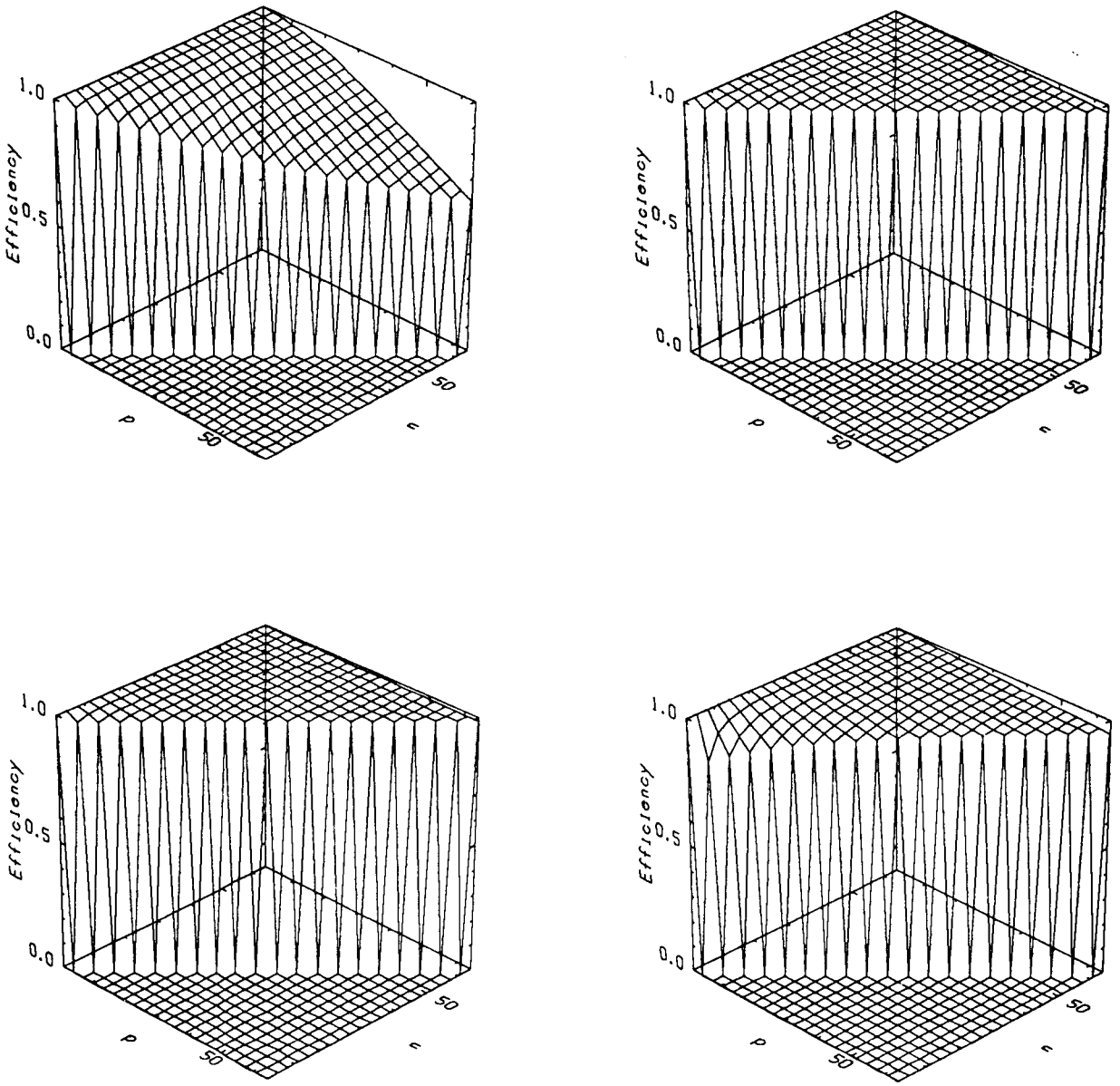

Figure 2: Surface plots of the parallel efficiency of the relaxation-based nonlinear elliptic solver on the methane-air axisymmetric jet diffusion flame problem for stripwise decompositions. The resolution of the domain in each dimension, n, runs between 2 and 64. The number of processors employed, p, runs between 1 and n. (a) upper left: bus-connected shared memory architecture with $b = 1$ ($a = 0$). (b) upper right: ring architecture with $a = 100$ and $b = 1$. (c) lower left: hypercube architecture with $a = 100$ and $b = 1$. (d) lower right: hypercube architecture with $a = 1000$ and $b = 1$.

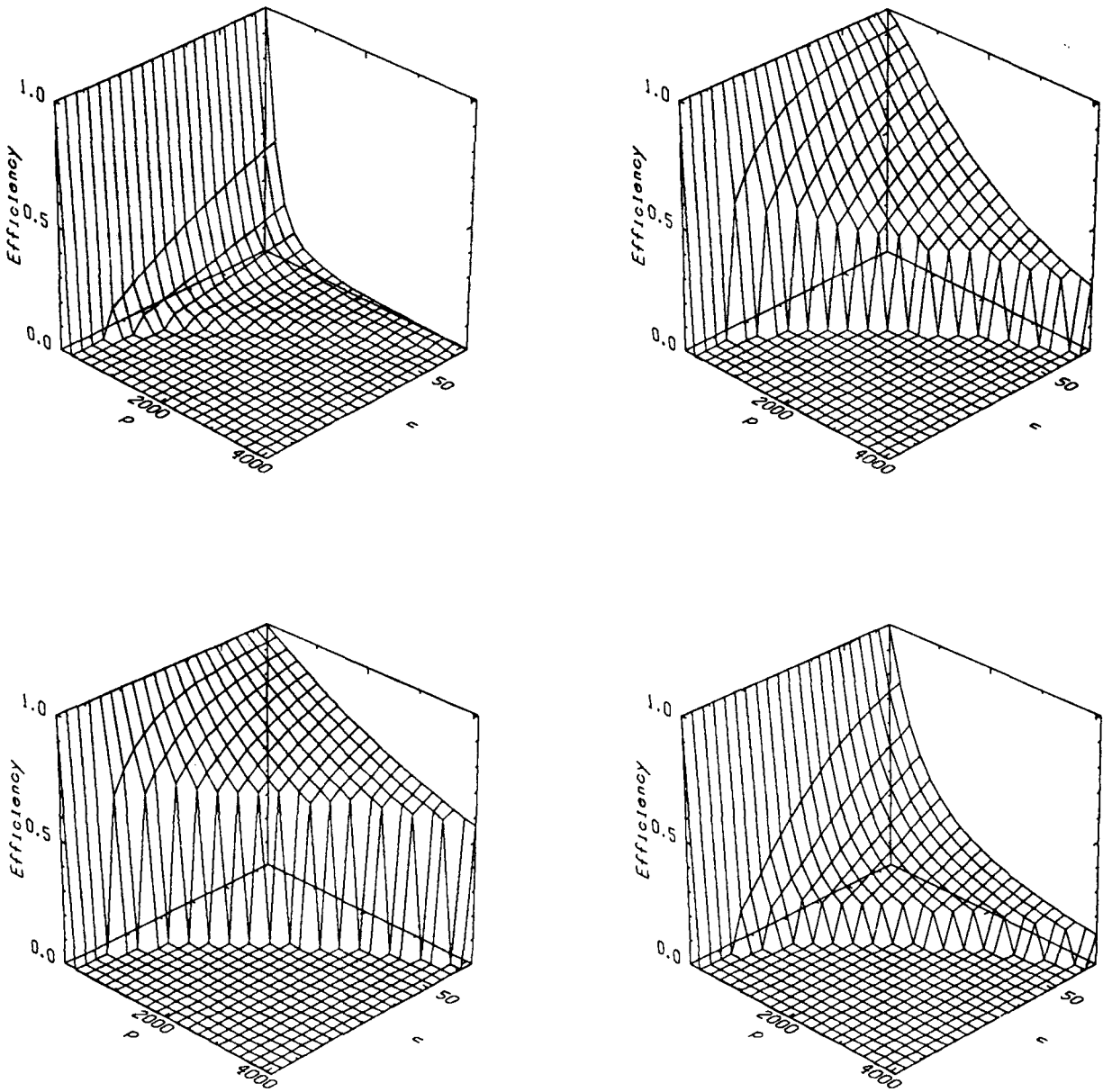

Figure 3: Surface plots of the parallel efficiency of the relaxation-based nonlinear elliptic solver on the methane-air axisymmetric jet diffusion flame problem for boxwise decompositions. The resolution of the domain in each dimension, n, runs between 2 and 64. The number of processors employed, p, runs between 1 and n^2. (a) upper left: bus-connected shared memory architecture with $b = 1$ ($a = 0$). (b) upper right: mesh architecture with $a = 100$ and $b = 1$. (c) lower left: hypercube architecture with $a = 100$ and $b = 1$. (d) lower right: hypercube architecture with $a = 1000$ and $b = 1$.

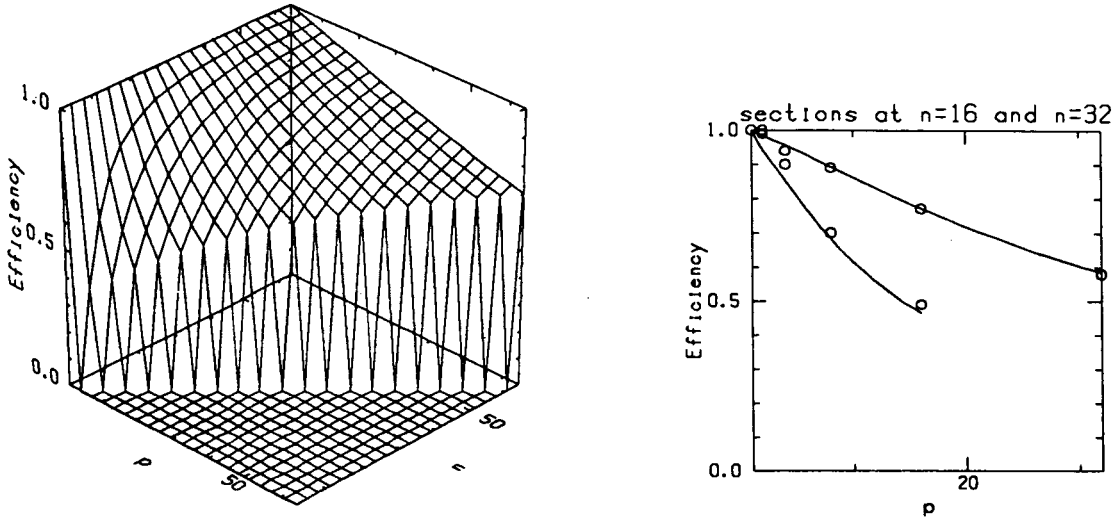

Figure 4: Theoretical (curves) and actual (plotted circles) parallel efficiency of the relaxation-based nonlinear elliptic solver on the model problem for stripwise decompositions on the Intel iPSC-5. (a) left: surface plot of theoretical efficiency over $2 \leq n \leq 64$, $1 \leq p \leq n$. (b) right: graph comparing theoretical and actual efficiency vs. p at $n = 16$ and $n = 32$.

Standard Bibliographic Page

1. Report No. NASA CR-178274 ICASE Report No. 87-21	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle ANALYSIS OF A PARALLELIZED NONLINEAR ELLIPTIC BOUNDARY VALUE PROBLEM SOLVER WITH APPLICATION TO REACTING FLOWS		5. Report Date April 1987 6. Performing Organization Code
7. Author(s) David E. Keyes, Mitchell D. Smooke		8. Performing Organization Report No. 87-21
9. Performing Organization Name and Address Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665-5225		10. Work Unit No. 11. Contract or Grant No. NAS1-18107
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		13. Type of Report and Period Covered Contractor Report 14. Sponsoring Agency Code 505-90-21-01

15. Supplementary Notes

Langley Technical Monitor: J. C. South
Final Report
Additional support provided under contract AFOSR 85-0189.

Submitted to the 6th International Symposium on Computer Methods for Partial Differential Equations, IMACS Conference, June 1987

16. Abstract

A parallelized finite difference code based on Newton's method for systems of nonlinear elliptic boundary value problems in two dimensions is analyzed in terms of computational complexity and parallel efficiency. An approximate cost function depending on 15 dimensionless parameters (including discrete problem dimensions, convergence parameters, and machine characteristics) is derived for algorithms based on stripwise and boxwise decompositions of the domain and a one-to-one assignment of the strip or box subdomains to processors. The sensitivity of the cost function to the parameters is explored in regions of parameter space corresponding to model small-order systems with inexpensive function evaluations and also a coupled system of nineteen equations with very expensive function evaluations (a reacting flow model of engineering interest which motivates the work). The algorithm has been implemented on the Intel Hypercube, and some experimental results for the model problems with stripwise decompositions are presented and compared with the theory. In the context of computational combustion problems, multiprocessors of either message-passing or shared-memory type may be employed with stripwise decompositions to realize speedups of $O(n)$, where n is mesh resolution in one direction, for reasonable n. To realize speedups of $O(n^2)$, the total number of mesh points, only hypercubes appear attractive. These results must be qualified by hardware assumptions, including sufficient local memory per processor to hold all of the data defined on the associated subdomain, and selection of machine parameters typical of presently commercially available components.

17. Key Words (Suggested by Authors(s)) nonlinear elliptic problems, Newton's method, parallel algorithms, computational fluid dynamics, combustion	18. Distribution Statement 64 - Numerical Analysis Unclassified - unlimited		
19. Security Classif.(of this report) Unclassified	20. Security Classif.(of this page) Unclassified	21. No. of Pages 28	22. Price A03

For sale by the National Technical Information Service, Springfield, Virginia 22161